Optimisation de la gestion sur disque des bibliothèques de programmes

J.Y. Babonneau, M. Carpentier, G. Morisset

INRIA

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P 105 - 78150 Le Chesnay
France
Tél. 954 90 20

Rapports de Recherche

N° 36

OPTIMISATION DE LA GESTION SUR DISQUES DES BIBLIOTHÈQUES DE PROGRAMMES

Jean-Yves BABONNEAU
Michel CARPENTIER
Gérard MORISSET

Septembre 1980

OPTIMISATION DE LA GESTION SUR-DISQUE DES BIBLIOTHEQUES DE PROGRAMMES

Jean-Yves BABONNEAU, Michel CARPENTIER, Gérard MORISSET

Résumé

Devant l'importance des problèmes posés par la gestion des bibliothèques, constituées de milliers d'éléments prédéfinis représentant un investissement d'intérêt général, nous avons commencé par en visualiser la structure sous la forme d'une représentation matricielle des liens inter-modules.

Nous avons poursuivi l'étude de cette structure par la mise en oeuvre d'algorithmes permettant d'y détecter certaines anomalies : cycles, trop grande interconnexion des modules, etc...

Nous avons enfin conçu et réalisé un système général et automatique susceptible d'augmenter la localité des références aux sous-programmes sur disque. Une réduction de 50 à 70% des accès disque a été obtenue.

L'ensemble du logiciel réalisé fait partie du système OPALE, qui a été mis à son catalogue par CII-HB en mai 1980.

Abstract

The management of program libraries which can include thousands of elements involves great problems. In view of this, the first thing we did was to display the structure of these libraries by using a module inter-relationship matricial representation.

We furthered the study of this structure by designing algorithms in order to detect some of its anomalies : cycles, too strong module interconnection, etc...

We finally conceived and realized a general automatic system to improve program reference locality on disk. A reduction of 50 to 70% of disk access was obtained.

All of the software realized forms part of the OPALE system, which was included by CII-HB in its catalogue in may 1980.

INTRODUCTION

Les systèmes d'exploitation sont constitués d'un ensemble complexe de programmes destinés à :

- optimiser la gestion des ressources de l'ordinateur,
- faciliter la tâche du programmeur.

C'est ainsi qu'il existe :

- des modules destinés à servir d'interface entre le programme utilisateur et le logiciel propre à l'ordinateur,
- des modules répondant à des problèmes généraux souvent rencontrés : fonctions mathématiques, résolution de problèmes économiques, etc ...

Ces modules sont en général regroupés dans des bibliothèques fréquemment référencées par les utilisateurs, souvent à leur insu.

Ces bibliothèques représentent un investissement important par le nombre et la complexité des modules qu'elles contiennent. Ce sont des objets structurés difficiles à maîtriser, car les relations entre leurs modules peuvent devenir très complexes. Leurs implantations physiques entraînent souvent des pertes importantes d'espace disque et en alourdissent l'accès.

Nous décrivons dans ce rapport un outil permettant de :

- connaître la fréquence d'utilisation des divers modules,
- déterminer la structure des bibliothèques,
- diminuer leur encombrement et leur coût d'accès en les restructurant.

Nous mettrons en oeuvre cette restructuration au moyen d'un cycle d'adaptation composé de trois phases :

- mesure,
- analyse,
- adaptation.

Au niveau de l'analyse, nous montrerons qu'il existe une similitude entre notre problème et celui de l'optimisation des programmes en milieu paginé. Ceci nous permettra d'adapter à nos objectifs les algorithmes mis en oeuvre dans le projet OPALE (BAB77).

CHAPITRE 1 : LES BIBLIOTHEQUES DE SOUS-PROGRAMMES

1.1 Utilisation des bibliothèques de sous-programmes dans la chaîne de traduction

La chaîne de traduction des programmes est constituée des étapes du passage du langage source au programme exécutable. La complexité et la taille des programmes imposent leur décomposition en éléments simples. La plupart des chaînes de traduction autorisent donc la compilation séparée des programmes. Elles se décomposent alors en deux étapes (figure 1) :

- des compilateurs ou assembleurs transforment n programmes sources en n programmes écrits dans un langage intermédiaire commun,
- un éditeur de liens réunit ces n programmes en un seul programme exécutable (PRE72).

Certaines facilités offertes par les langages évolués dépassent actuellement le cadre des instructions câblées ou microprogrammées : entrées/sorties, fonctions mathématiques, etc. On fait donc appel à des sous-programmes qui seront adjoints automatiquement au programme principal, en général au moment de l'édition de liens. Ces sous-programmes sont stockés dans des bibliothèques étroitement liées aux compilateurs et représentent souvent un travail aussi considérable que le compilateur lui-même.

1.2 Organisation et méthode d'accès aux bibliothèques sous SIRIS8

Notre environnement de travail a été celui du système SIRIS8 version C10 (à mémoire virtuelle paginée) de l'ordinateur CII-HB IRIS80.

1.2.1 Structure d'une bibliothèque

Le fichier utilisé pour stocker la bibliothèque est un fichier partitionné (SGF 75). On peut distinguer 2 parties dans un tel fichier (figure 2) :

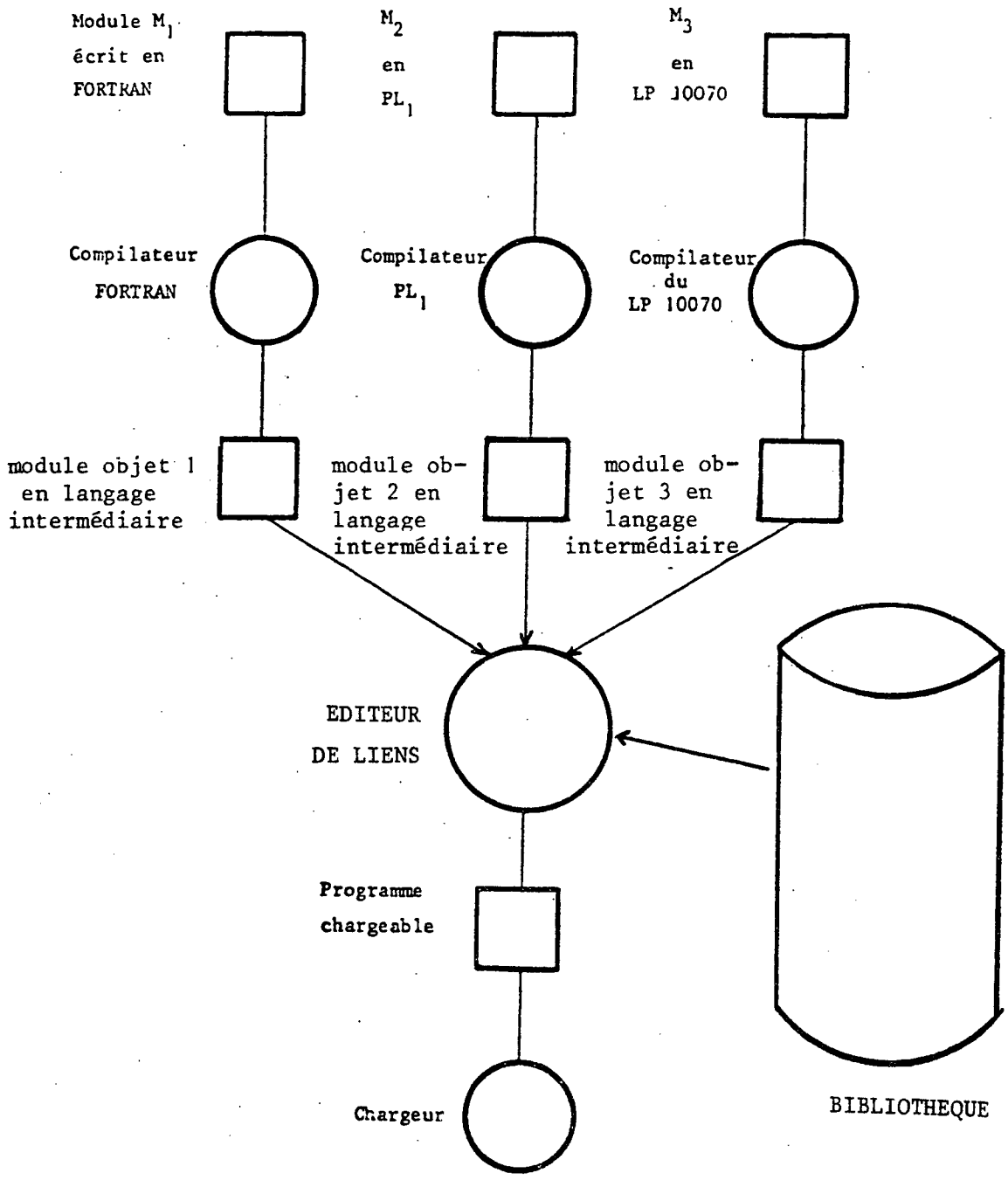
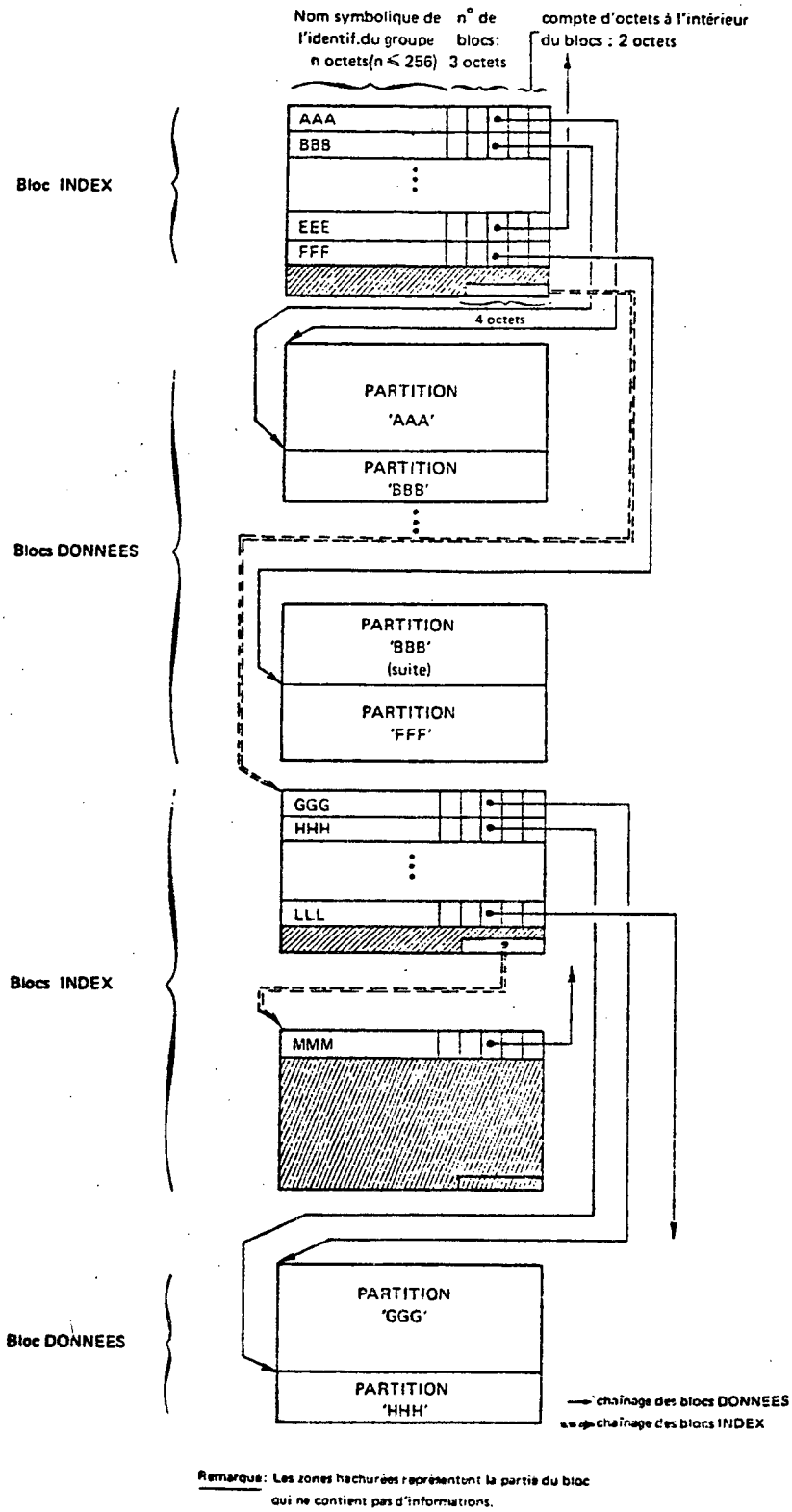


Schéma de la chaîne de traduction d'un programme constitué de trois modules écrits en FORTRAN, PL1, LP 10070.

Chaîne de traduction

Figure 1



Remarque: Les zones hachurées représentent la partie du bloc qui ne contient pas d'informations.

Organisation partitionnée

Figure 2

- l'index d'accès aux sous-programmes, qui est constitué de blocs chaînés. Les clefs sont stockées dans l'ordre alphabétique.
- les sous-programmes eux-mêmes, qui occupent chacun une partition constituée de blocs consécutifs.

L'alignement systématique des sous-programmes sur une frontière de bloc peut provoquer une perte de place importante : jusqu'à 40% dans un exemple réel, car leur taille est souvent très inférieure à un bloc.

1.2.2 Edition de liens et accès aux bibliothèques

Le LINK, éditeur de liens standard sous SIRIS8, effectue un certain nombre de passes pour résoudre les références externes au cours de l'édition de liens. Chaque passe peut se décomposer en trois étapes :

- prise en compte de la première référence non satisfaite,
- positionnement dans la bibliothèque à l'emplacement du module qui satisfait cette référence,
- édition des liens de ce module avec le programme principal.

La structure de l'index des fichiers partitionnés fait que l'on dépense rapidement autant d'efforts pour localiser un sous-programme que pour le charger en mémoire.

Le RELIEUR, développé à l'INRIA dans le cadre du projet OPALE, utilise dans le cas particulier de l'accès aux bibliothèques standard SIRIS8, une méthode plus performante : il tient compte de l'ordre alphabétique des clefs, ce qui lui permet de résoudre plusieurs références non satisfaites à chaque passe.

1.3 Dynamique des bibliothèques de sous-programmes

1.3.1 Nature des mesures prélevées

Nous prélevons les mesures au niveau de chaque édition de liens. Une mesure se présente sous la forme d'une liste des noms des sous-programmes référencés par le programme dans la bibliothèque.

1.3.2 Résultats

Les résultats présentés ci-dessous ont été obtenus à partir de la bibliothèque LIBREL utilisée à l'INRIA ; LIBREL regroupe les bibliothèques FORTRAN, ALGOL, COBOL, BENSON et TEKTRONIX, c'est-à-dire les bibliothèques utilisées le plus couramment au centre de calcul de l'INRIA.

La figure 3 représente la courbe de distribution de probabilité du nombre de sous-programmes appelés par édition de liens. Le nombre moyen de sous-programmes appelés est 28, l'écart type est de 5,78.

La figure 4 représente la courbe de distribution de probabilité de la taille cumulée des sous-programmes référencés à chaque édition de liens.

Les tailles sont exprimées en % de la taille totale utile de la bibliothèque. La taille moyenne est de 17%, l'écart type est de 3,51%.

La figure 5 représente les fréquences d'utilisation des sous-programmes.

Il apparaît que :

- un nombre important (70%) d'éléments n'est jamais utilisé,
- un petit nombre d'entre eux (11%) représente 85% de l'utilisation de la bibliothèque.

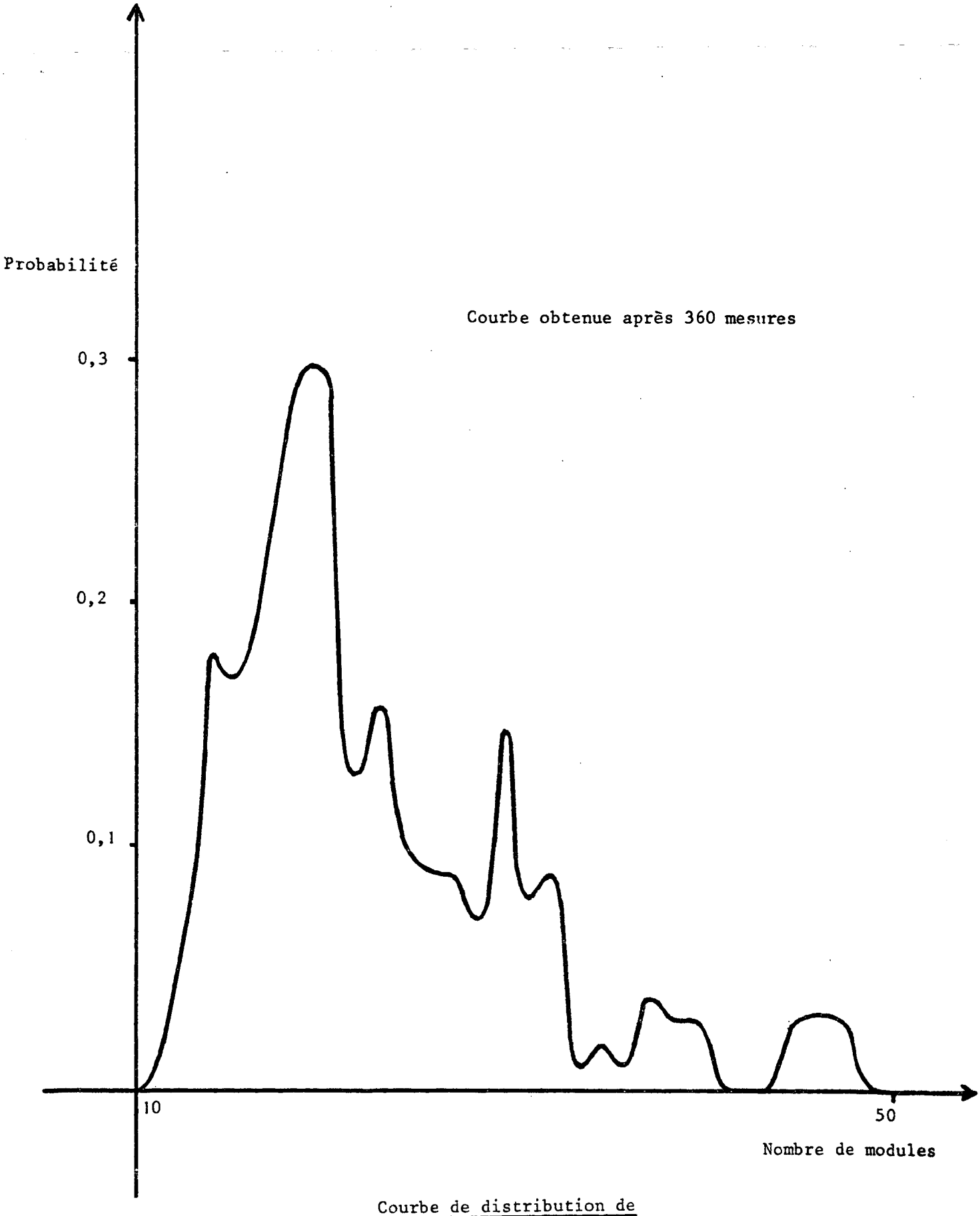
Nous en concluons qu'il existe un "noyau" qui regroupe un petit nombre de sous-programmes fortement référencés.

Des résultats similaires ont été obtenus pour d'autres périodes de mesure : voir figures 6 et 7. Ces résultats nous permettent en outre de vérifier une certaine stabilité du noyau dans le temps.

1.4 Conclusion

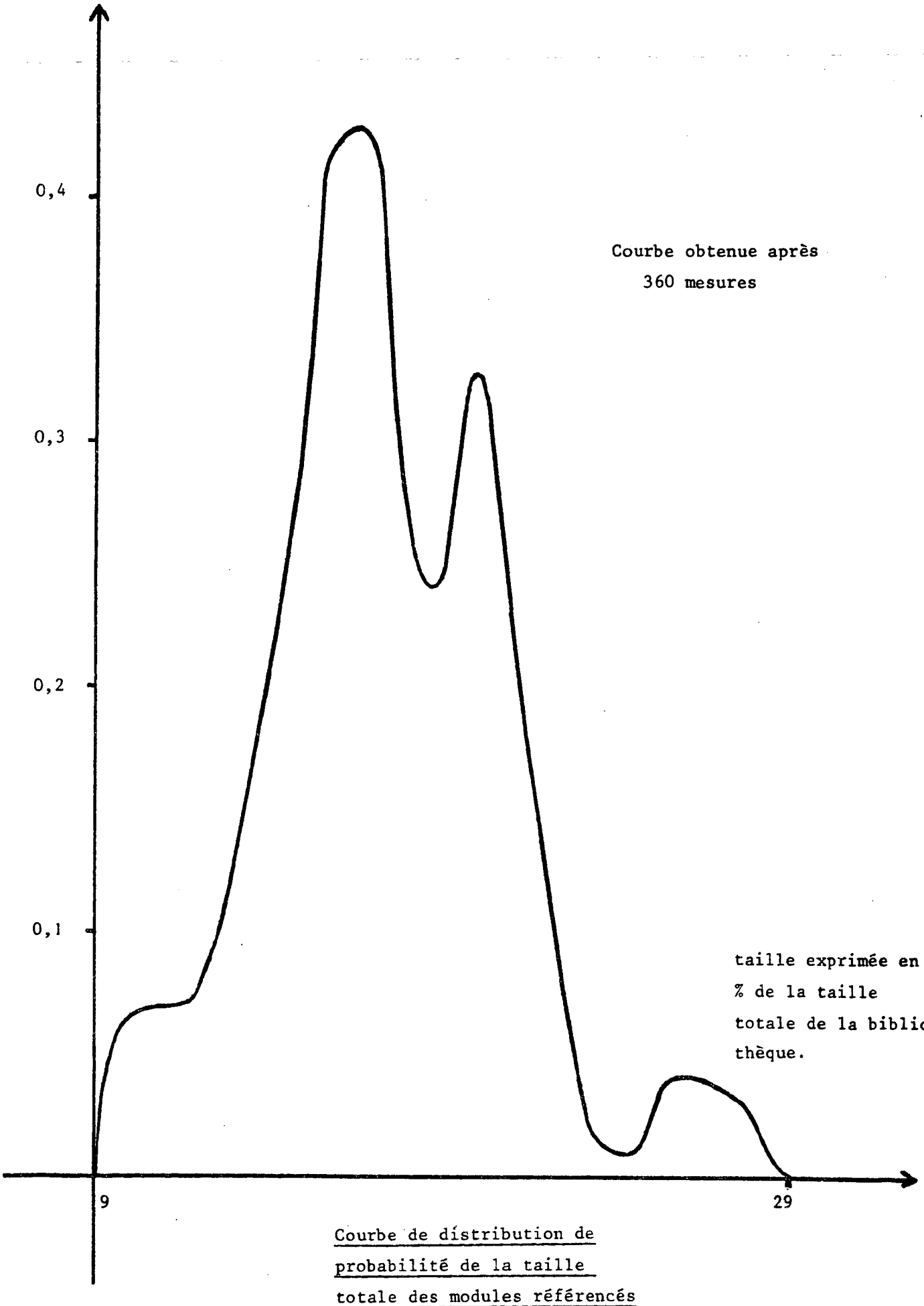
L'existence d'un noyau stable suggère une possibilité d'amélioration des bibliothèques :

- extraire ce noyau,
- lui adjoindre une méthode d'accès plus performante.



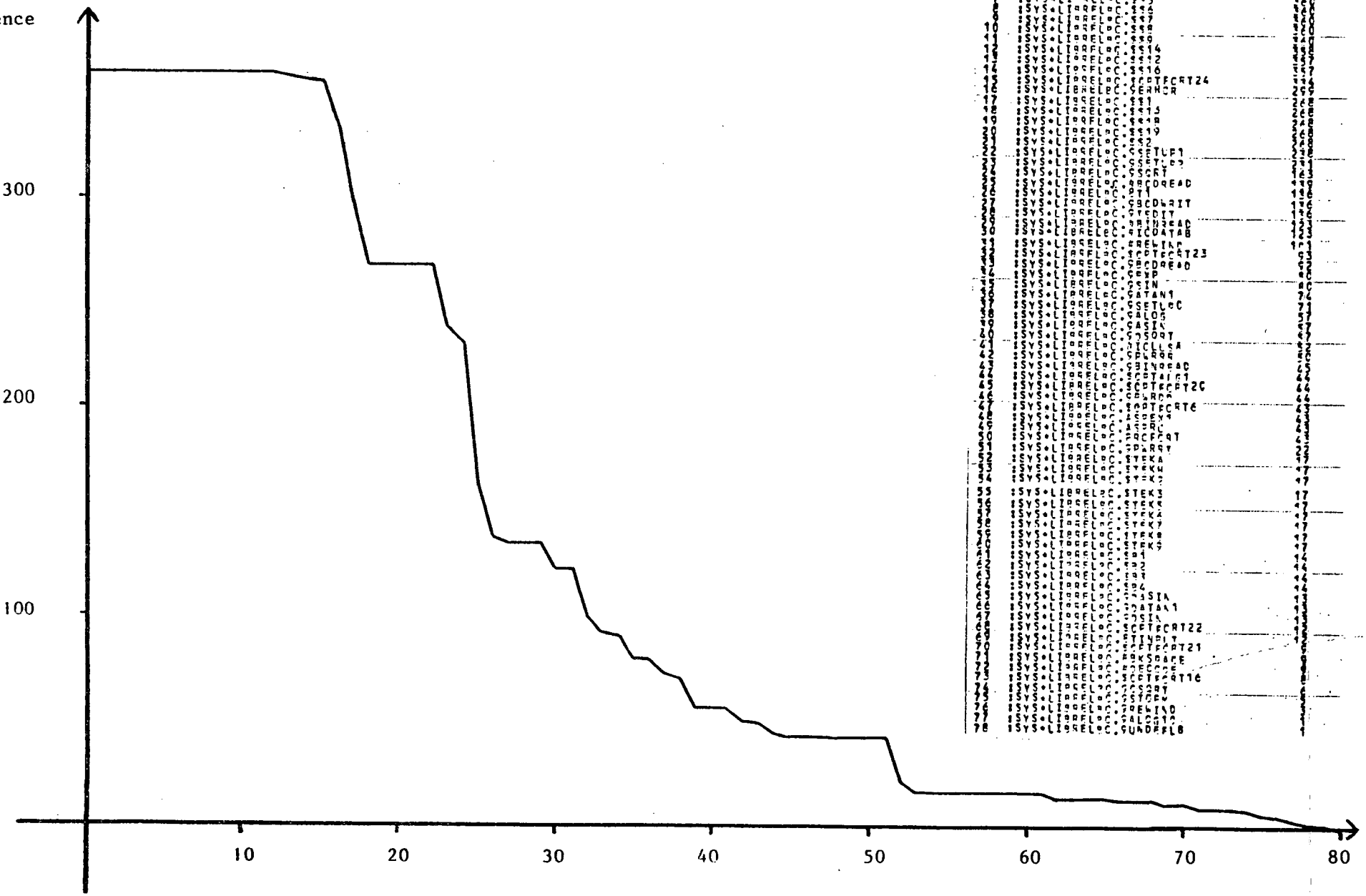
Courbe de distribution de probabilité du nombre de modules référencés

Figure 3



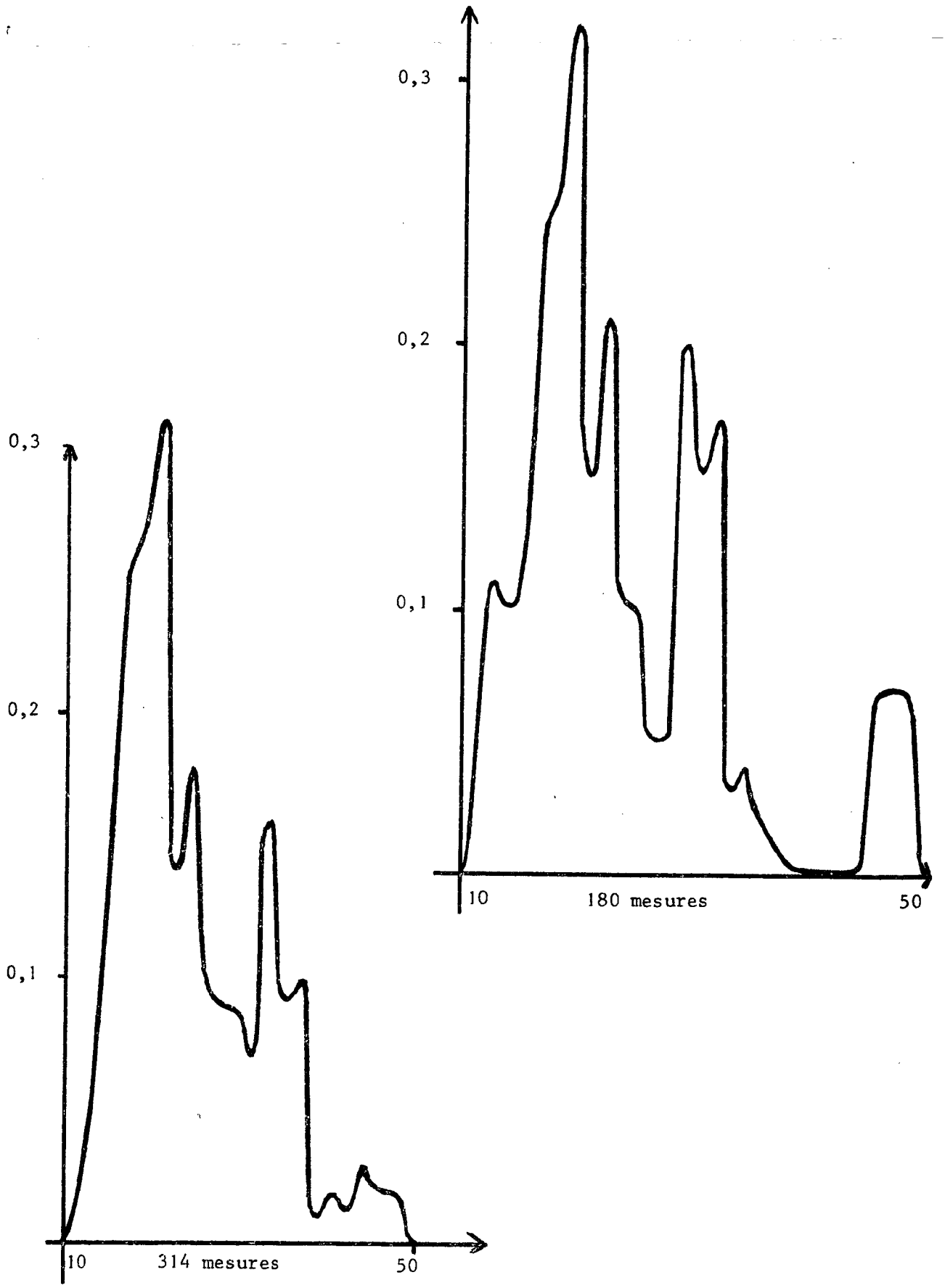
Courbe de distribution de probabilité de la taille totale des modules référencés

Figure 4



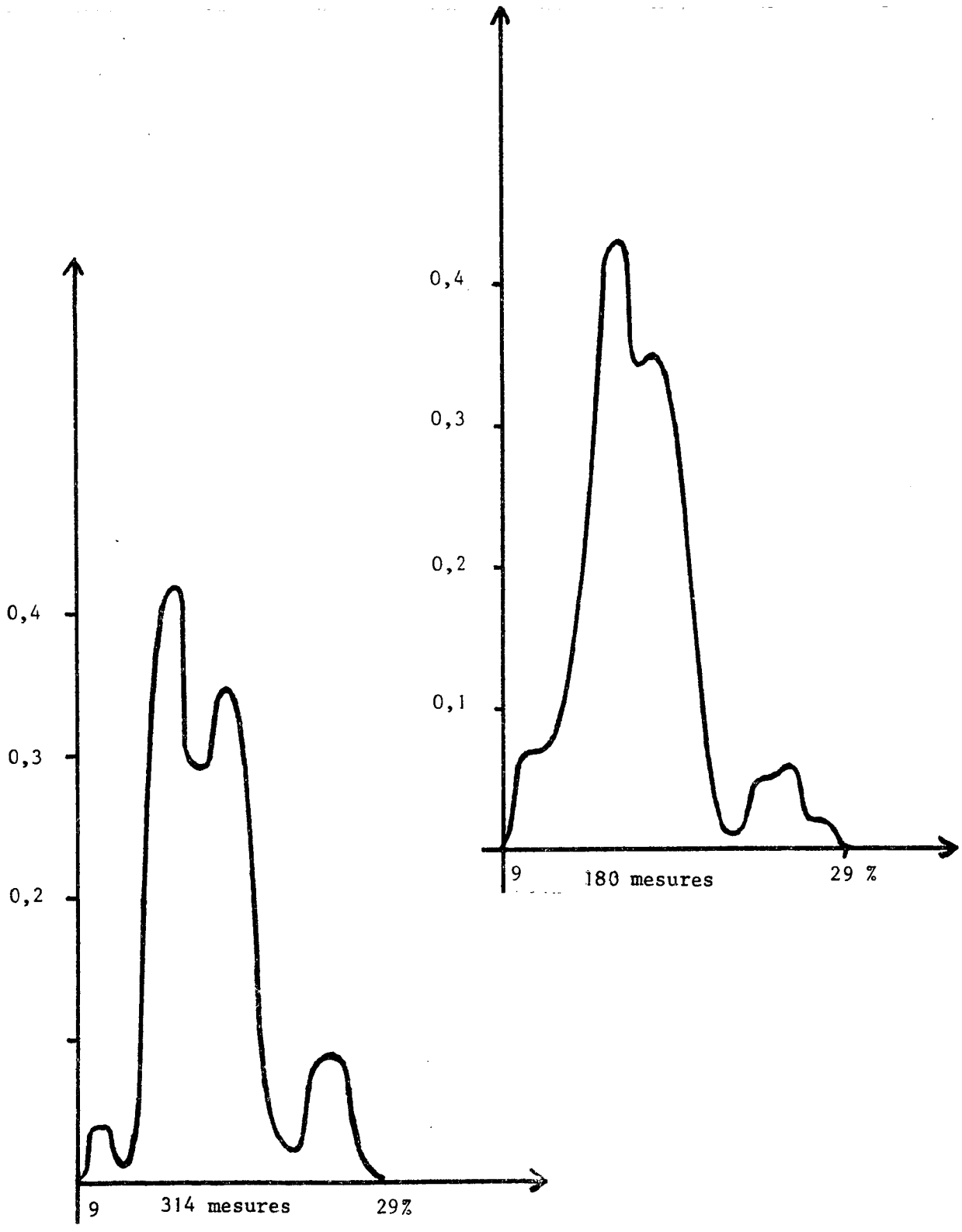
Fréquence d'utilisation des modules

Figure 5



Courbes de distribution de probabilité du nombre de modules référencés

Figure 6



Courbe de distribution de probabilité de la taille totale des modules

Figure 7

CHAPITRE 2 : OPTIMISATION DES BIBLIOTHEQUES

2.1 Relation entre les bibliothèques de sous-programmes et la mémoire paginée

2.1.1 Principe de la mémoire virtuelle

Le but visé par la notion de mémoire virtuelle est d'offrir aux programmes un espace plus grand que la mémoire réelle disponible. La mémoire virtuelle et la mémoire réelle sont découpées en blocs de taille fixe appelés "pages". Les pages virtuelles sont amenées depuis le disque en mémoire centrale selon les besoins de l'exécution du programme. Lorsqu'une page référencée est absente de la mémoire centrale (défaut de page), il faut rendre disponible l'une des pages réelles. On utilise alors un algorithme de remplacement pour désigner la page qui sera renvoyée sur disque, si elle a été modifiée.

Le bon fonctionnement de la mémoire virtuelle repose sur un taux de pagination faible. Si un programme utilise constamment toutes ses pages, ou si l'espace réel est trop petit, le taux de pagination devient prohibitif et l'efficacité du système s'effondre.

2.1.2 Principe d'adaptation des programmes au milieu paginé

Il s'agit de diminuer le nombre de défauts de pages engendrés par l'exécution du programme.

Dans ce but, on regroupe dans de mêmes pages les informations qui sont référencées dans le même intervalle de temps.

Cette approche peut être suivie pour des programmes constitués d'unités logiques ayant une taille moyenne très inférieure à une page.

Les corps des procédures et les zones de données générés par la plupart des traducteurs forment de tels ensembles, ou "sections", et peuvent être manipulés facilement (MOR75).

2.1.3 Comparaison de l'utilisation d'une bibliothèque et du comportement d'un programme en milieu paginé

Similitudes "statiques" :

La bibliothèque est un ensemble de sous-programmes, le programme est un ensemble de sections. Les sous-programmes de la bibliothèque sont stockés sur des quantums d'espace disque, les sections du programme le sont sur des quantums de mémoire. Il existe dans les deux cas des liens statiques entre les objets, sous-programmes ou sections. Ces deux cas correspondent à une projection d'un milieu logique constitué d'unités de tailles quelconques dans un milieu physique quantifié.

Similitudes "dynamiques" :

Pour les sous-programmes comme pour les sections, un petit nombre d'éléments concentre la grande majorité des références (BAB77).

Similitude de problème :

L'exécution d'un programme en milieu paginé et l'accès aux modules d'une bibliothèque impliquent le chargement en mémoire de quantums d'espace disque. Pour les bibliothèques comme pour les programmes paginés, le quantum chargé peut contenir en outre des éléments qui seront utilisés par la suite : dans un avenir proche dans le cas du programme, au cours de la même édition de liens dans le cas de la bibliothèque.

2.1.4 Utilisation du système OPALE

Nous venons de montrer que les problèmes de gestion d'une mémoire paginée et d'une bibliothèque sont similaires : tous deux sont en effet des problèmes d'échange entre deux niveaux de mémoire quantifiées. Ceci nous amènera à utiliser les solutions mises en oeuvre dans le système OPALE (ACH76).

2.2 Restructuration des bibliothèques de sous-programmes

Nous proposons une méthode de restructuration combinant trois principes (figure 8) :

- nous isolons dans la bibliothèque les éléments les plus utilisés (Noyau),
- nous mettons en oeuvre une méthode d'accès au Noyau plus simple et plus rapide,
- nous appliquons sur l'espace du Noyau des algorithmes de regroupement développés dans le projet OPALE.

2.2.1 Séparation du Noyau

L'ensemble des sous-programmes d'une bibliothèque peut se diviser en deux parties :

- les sous-programmes fortement utilisés (Noyau), sur disque rapide par exemple,
- les sous-programmes faiblement utilisés sur disque lent.

En dehors de toute considération de vitesse du disque, il est intéressant de séparer le Noyau du reste de la bibliothèque pour n'appliquer que sur lui des méthodes d'optimisation qui peuvent être coûteuses (2.2.2 et 2.2.3).

Deux problèmes restent posés :

- savoir estimer quels sont les éléments constituant le Noyau. Ce problème sera résolu en 3.1.2 et 3.1.3.
- risque d'oscillation entre le Noyau et le reste de la bibliothèque au cours de l'édition de liens, par suite des références logiques entre sous-programmes. Une solution sera apportée par l'introduction de la notion de RACINE en 3.4.

2.2.2 Modification de la méthode d'accès

Nous avons vu (1.2.1 et 1.2.2) que l'organisation partitionnée se révèle coûteuse en place et en accès disque. Par conséquent nous avons implémenté une nouvelle méthode d'accès mieux adaptée en utilisant une recherche de clefs plus performante par "hash-code".

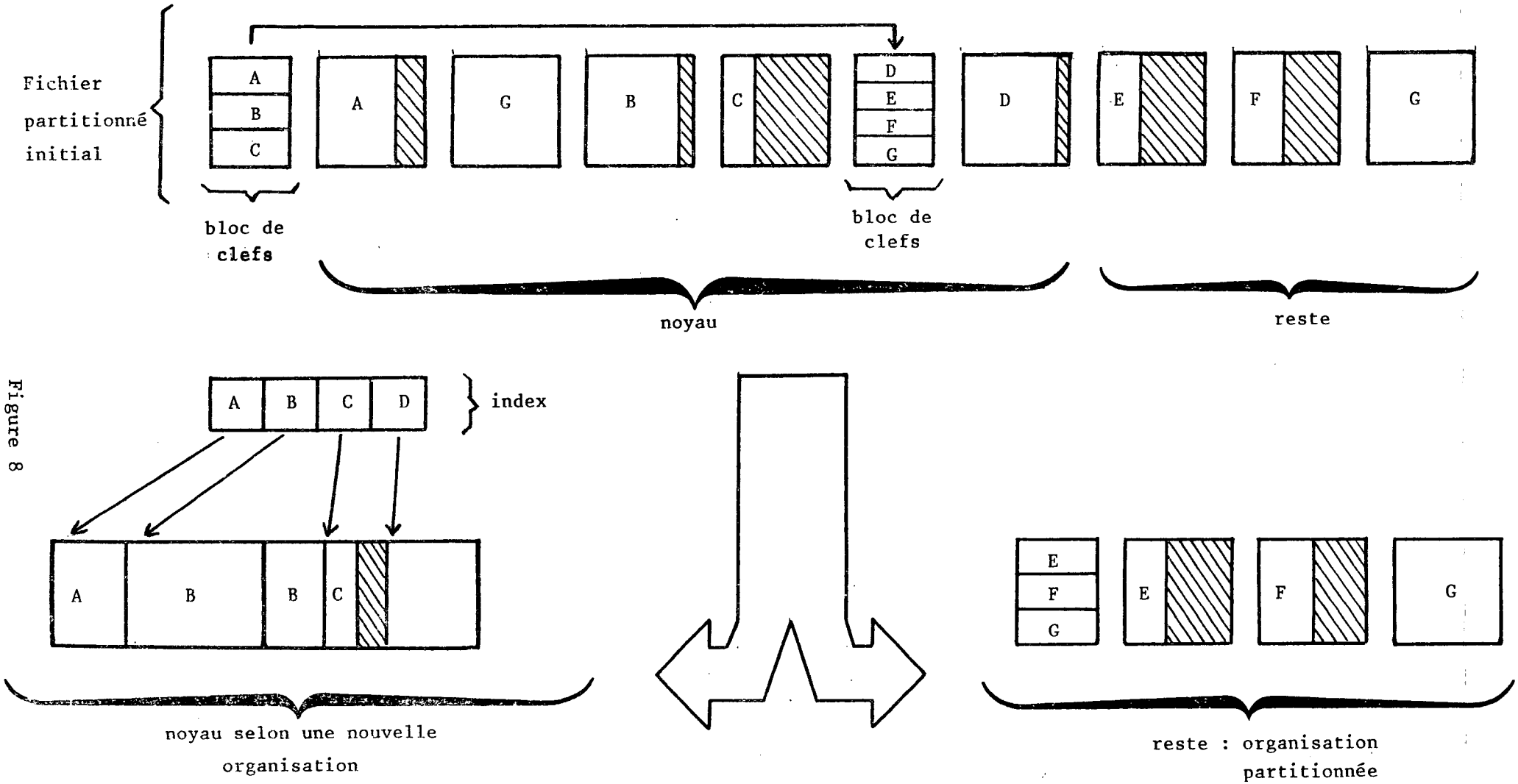


Figure 8

Méthode de restructuration proposée

2.2.3 Regroupements par affinités

Nous faisons appel ici à la similitude qui existe entre les bibliothèques et les programmes paginés (2.1.3).

Il est clair que l'accès au "coup par coup", qui est la méthode utilisée par le LINK, ne permet pas de minimiser le nombre d'accès disque dûs aux chargements des sous-programmes de la bibliothèque. Supposons par contre l'existence de "commandes groupées" : la réponse à une telle commande est un ensemble de sous-programmes. Si ces sous-programmes sont rassemblés par affinités dans de mêmes quantums, c'est-à-dire par tendance à être référencés ensembles, on peut minimiser le nombre de quantums à charger.

La connaissance a priori de tous les sous-programmes nécessaires et suffisante est une condition indispensable au bon fonctionnement de cette méthode. Cette connaissance est concrétisée par la notion de RACINE (3.4).

2.2.4 Enchaînement des étapes de la restructuration

La restructuration d'une bibliothèque est analogue à un cycle de restructuration d'un programme paginé. Cette restructuration se déroule en trois étapes (figure 9) :

- mesure par le BIBLIOTHECAIRE du comportement de la bibliothèque (le BIBLIOTHECAIRE contrôle l'accès aux bibliothèques).
- analyse des mesures, choix du Noyau et de son implémentation,
- prise en compte de cette implantation et création de la bibliothèque restructurée.

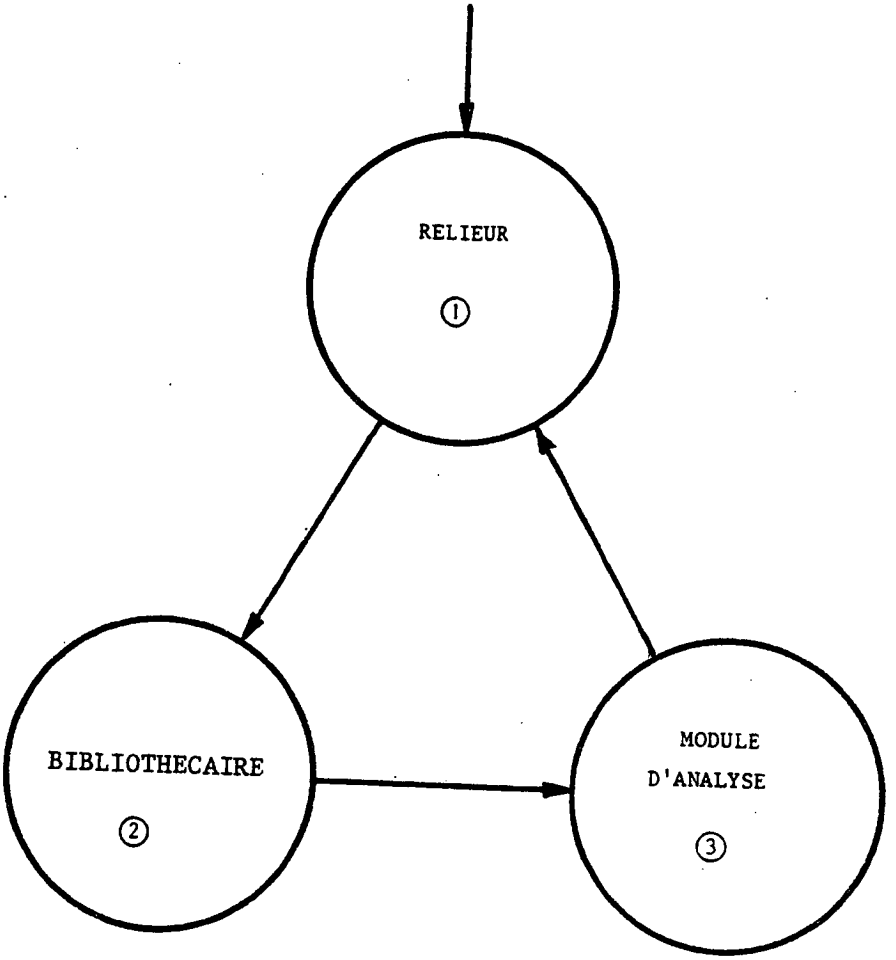
2.3 Conclusion

Nous possédons désormais tous les éléments qui nous permettent de mettre en oeuvre la restructuration des bibliothèques :

- une organisation autorisant le déroulement en trois étapes de la restructuration (triangle d'adaptation),
- les algorithmes de regroupement du système OPALE.

De nombreux problèmes de mise en oeuvre restent à résoudre :

- encombrement du Noyau,
- prise de mesures,
- choix du meilleur algorithme (simulation).



Triangle d'auto-adaptation

figure 9

CHAPITRE 3 : MISE EN OEUVRE

3.1 Problèmes liés à la notion de Noyau

3.1.1 Problème de taille

La détermination de la taille du Noyau est laissée, pour des raisons de souplesse, à l'utilisateur. Le SIMULATEUR fournit à ce dernier une estimation des gains apportés par diverses tailles du Noyau.

3.1.2 Choix des sous-programmes contenus dans le Noyau

Notre démarche sera de maximiser la probabilité d'accès au noyau, puisque ce dernier a une structure privilégiée.

Les sous-programmes sont classés dans l'ordre décroissant de densité, la densité étant définie par :

densité = fréquence d'utilisation/taille.

Cet ordre et la taille allouée au Noyau déterminent les sous-programmes le constituant.

3.2 Prise de mesure

Nous avons choisi de donner aux mesures la forme d'une liste des sous-programmes référencés dans la bibliothèque à chaque édition de liens.

Cette forme de mesure garantit la conservation totale des informations, ce qui nous permettra d'effectuer des simulations significatives.

3.3 Le SIMULATEUR

Le but du SIMULATEUR est de calculer, à partir des mesures, le nombre réel d'accès disque qui auraient été effectués au cours de la période de mesure si les sous-programmes constituant le Noyau avaient eu telle ou telle implantation.

3.3.1 Simulation des regroupements par affinités

Les algorithmes du système OPALE permettent de calculer plusieurs implantations possibles du Noyau à partir des mesures prélevées sur la bibliothèque.

Le SIMULATEUR calcule à partir de ces mêmes mesures le coût en accès disque de chaque implantation. Nous sommes ainsi assurés de choisir la meilleure implantation parmi celles qui sont envisagées.

3.3.2 Simulation en vue de la détermination de la taille du Noyau

La méthode retenue est d'effectuer la simulation en fonction de la taille du Noyau sur une seule implantation des sous-programmes : ceux-ci sont disposés dans l'ordre des densités décroissantes, sans alignement sur les frontières de quantums. Nous avons choisi cette méthode pour deux raisons :

- l'expérience montre que cette disposition est proche de l'optimale calculée par les algorithmes sophistiqués,
- l'implantation reste fixe quelle que soit la taille du Noyau. Il est donc possible de faire toutes les simulations en même temps.

3.4 RACINE

La liste des références non satisfaites doit, en une seule consultation de la RACINE, être transformée en une liste de tous les sous-programmes de la bibliothèque nécessaires et suffisants à l'édition de liens. Ceci est obligatoire pour donner tout son intérêt au principe de regroupement par affinités des sous-programmes du Noyau.

La RACINE contiendra donc (figure 10) :

- une liste donnant pour chaque référence externe le sous-programme qui la définit,
- une matrice donnant pour chaque sous-programme la liste des sous-programmes auxquels il fait référence directement et indirectement (fermeture transitive de la matrice : figure 11).

3.5 Déroulement d'une restructuration

3.5.1 Création d'une RACINE mesurable et phase de mesure

Pendant la phase de mesure des blocs consécutifs contenant les mesures viennent s'ajouter à la fin de la RACINE. Cette dernière étant consultée par plusieurs utilisateurs simultanés, la mise à jour des mesures pose des problèmes de conflit d'accès que nous avons dû résoudre.

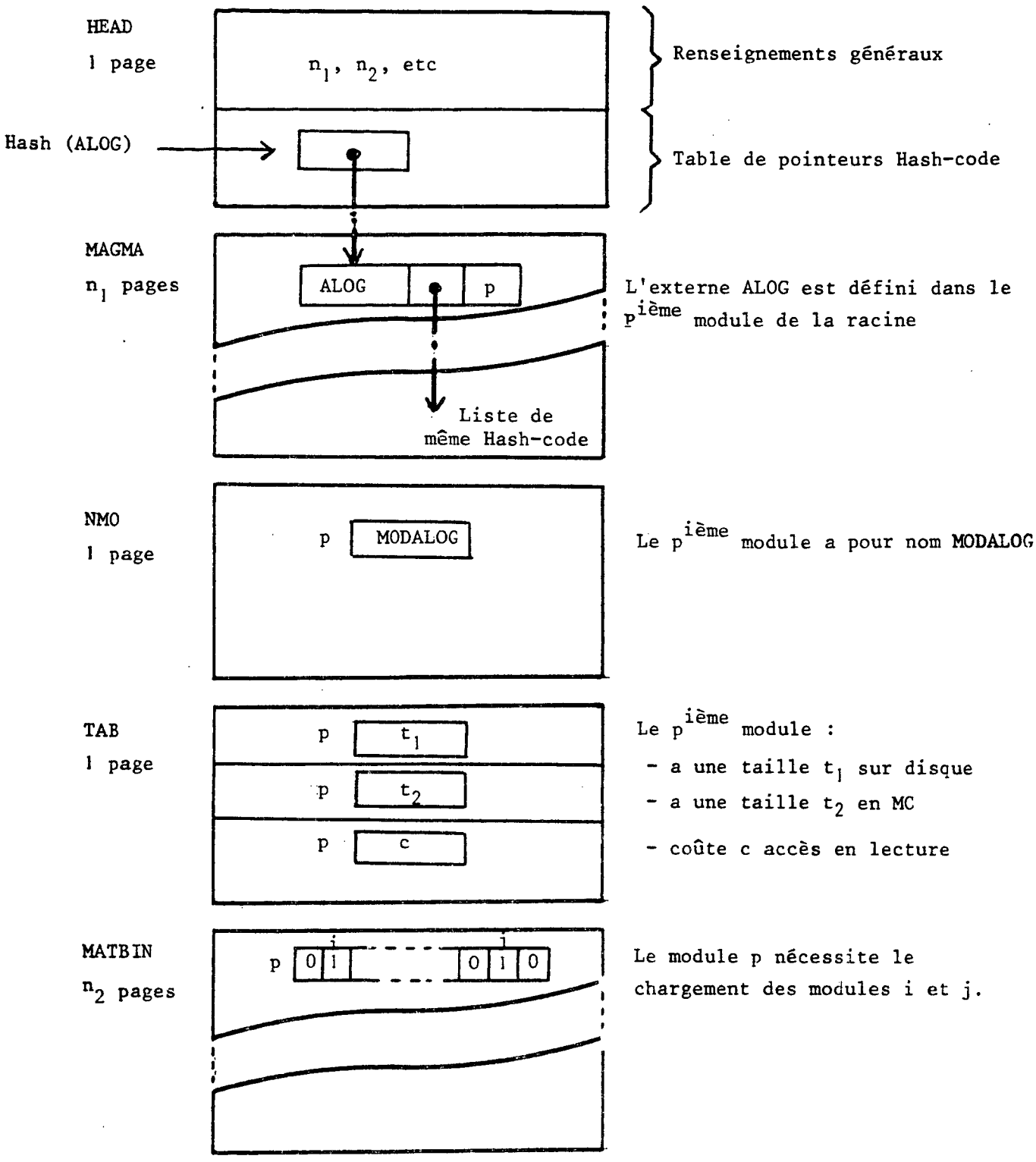
3.5.2 Visualisation de l'utilisation de la bibliothèque

L'analyse des mesures fournit d'abord trois courbes représentant l'utilisation des sous-programmes de la bibliothèque (voir 1.3.2) :

- courbe des fréquences d'utilisation,
- courbe de distribution de probabilité du nombre de sous-programmes référencés par édition de liens,
- courbe de distribution de probabilité de la taille des sous-programmes référencés par édition de liens.

Des simulations sur les mesures fournissent ensuite, en fonction de la taille du Noyau :

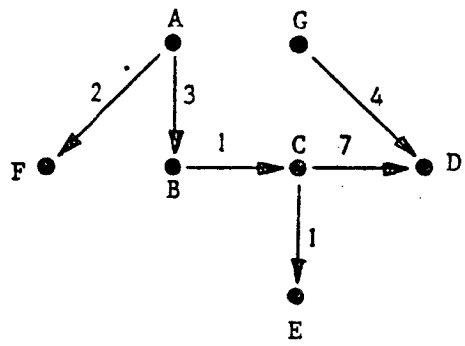
- des courbes des accès disque, (figure 12),
- une courbe du nombre d'octets transférés, (figure 13).



Structure d'une racine

Figure 10

A,B,C,D,E,F,G sont les modules de la bibliothèque. Le graphe des liens entre modules est le suivant :



Le module A fait 3 fois référence au module B qui fait 1 fois référence au module C e.t.c....

MATRICE DIRECTE MD

	A	B	C	D	E	F	G
A		3				2	
B			1				
C				7	1		
D							
E							
F							
G				4			

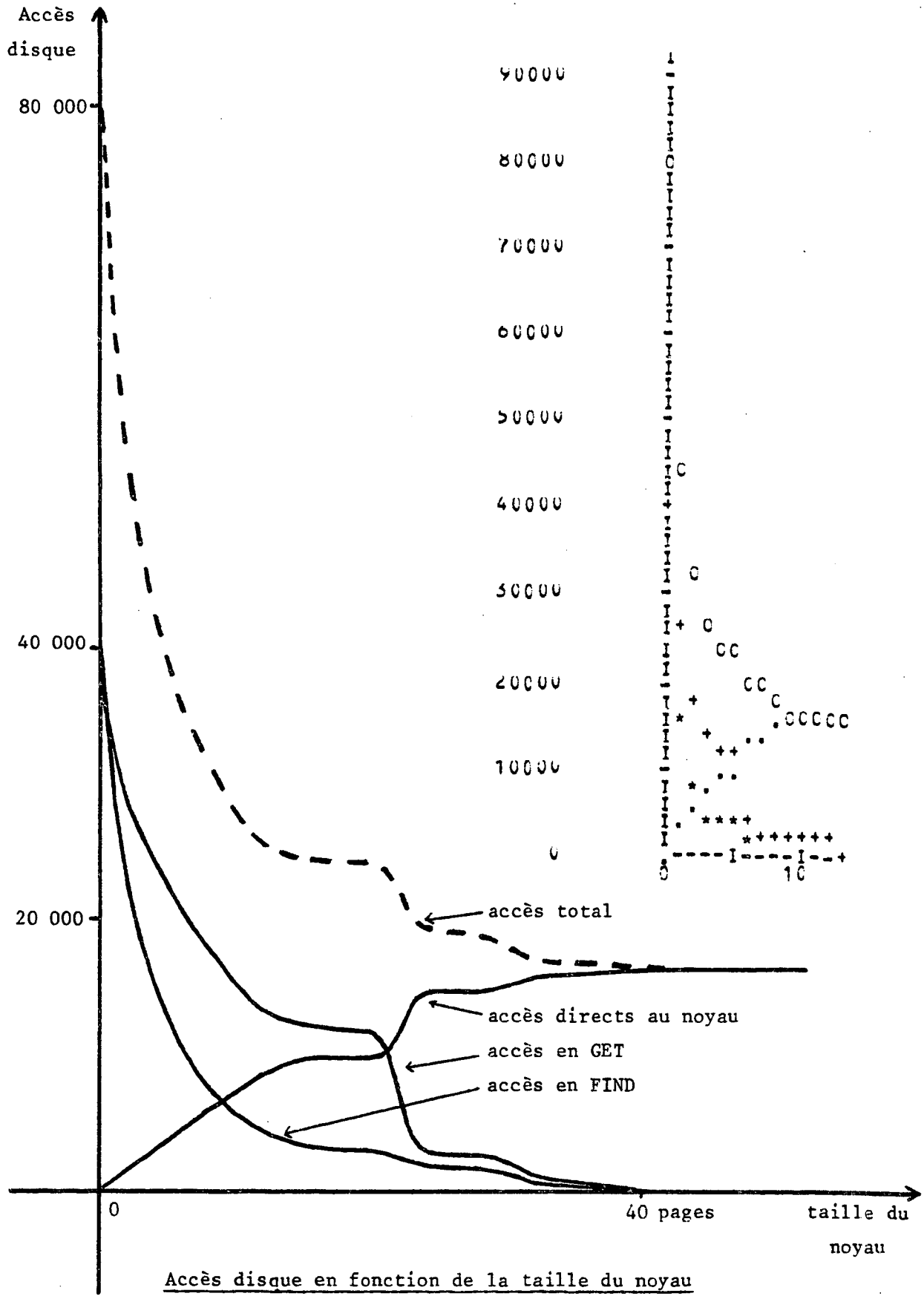
MATRICE INDIRECTE MI

	A	B	C	D	E	F	G
A	x	x	x	x	x	x	
B		x	x	x	x		
C			x	x	x		
D				x			
E					x		
F						x	
G				x			x

Fermeture transitive de la matrice

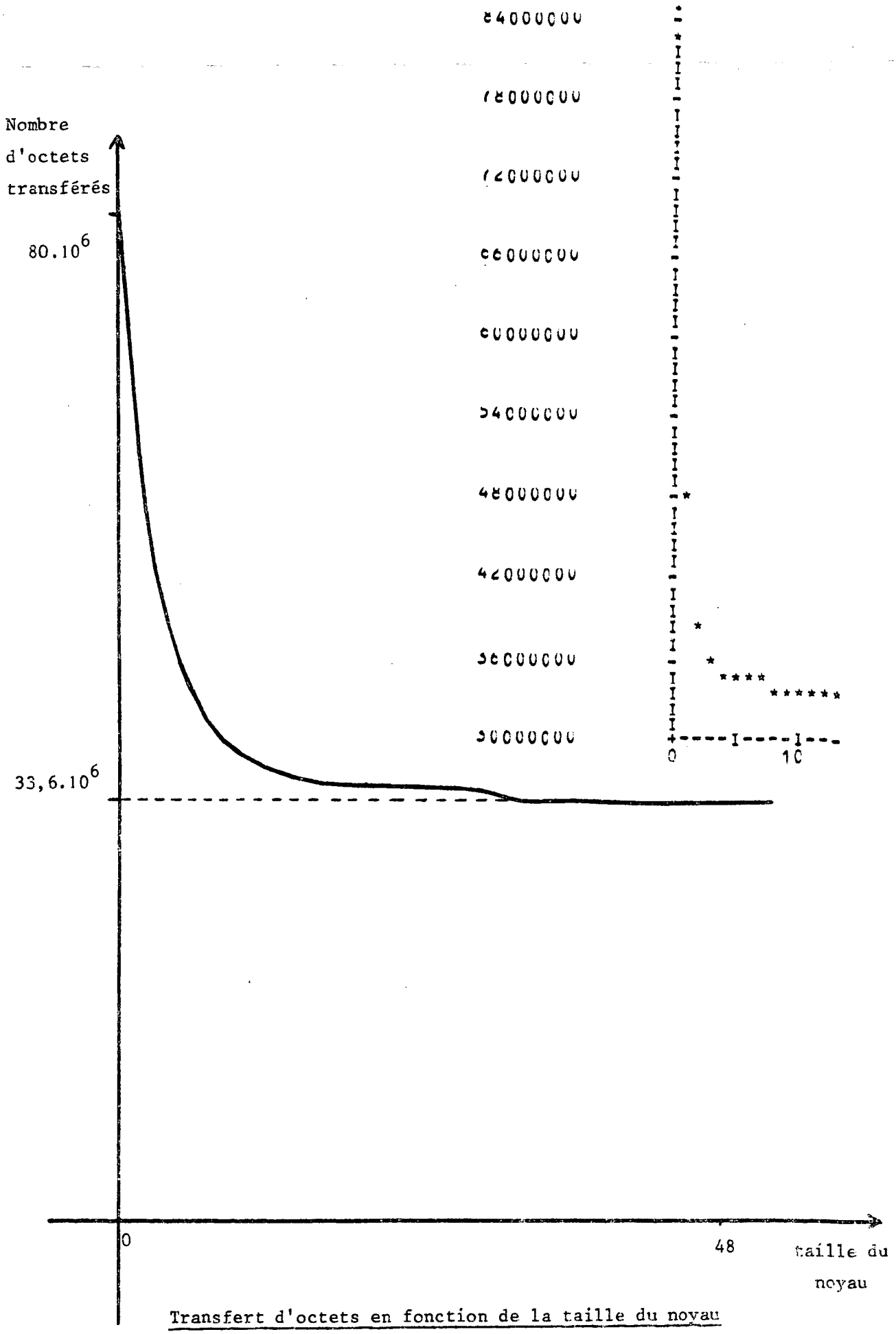
des références inter-modules

Figure 11



Accès disque en fonction de la taille du noyau

Figure 12



Transfert d'octets en fonction de la taille du noyau

Figure 13

3.5.3 Restructuration proprement dite

3.5.3.1 Délimitation du Noyau

Nous appliquons la méthode définie en 3.1.2. Remarquons que le Noyau peut contenir l'ensemble de la bibliothèque.

3.5.3.2 Simulation

Les mesures contenues dans la RACINE permettent aux algorithmes de regroupement, contenus dans l'ANALYSEUR, de calculer plusieurs implantations possibles des sous-programmes du Noyau.

Le SIMULATEUR détermine ensuite la meilleure.

Les algorithmes que nous avons utilisés sont au nombre de quatre :

- Algorithme PM : "du premier module" (ACH78),
- Algorithme CN : "de construction de Noyau" (ACH78),
- Algorithme CH : algorithme de classification hiérarchique,
- Algorithme ND : "de segmentation" (nuées dynamiques) (DID71).

Les versions de ces algorithmes sont celles qui sont utilisées sous le système OPALE pour l'adaptation des programmes au milieu paginé.

3.5.3.3 Adjonction du Noyau (figure 14)

La simulation a permis de déterminer l'implantation des sousprogrammes du Noyau. En fin de restructuration le Noyau est adjoint à la RACINE.

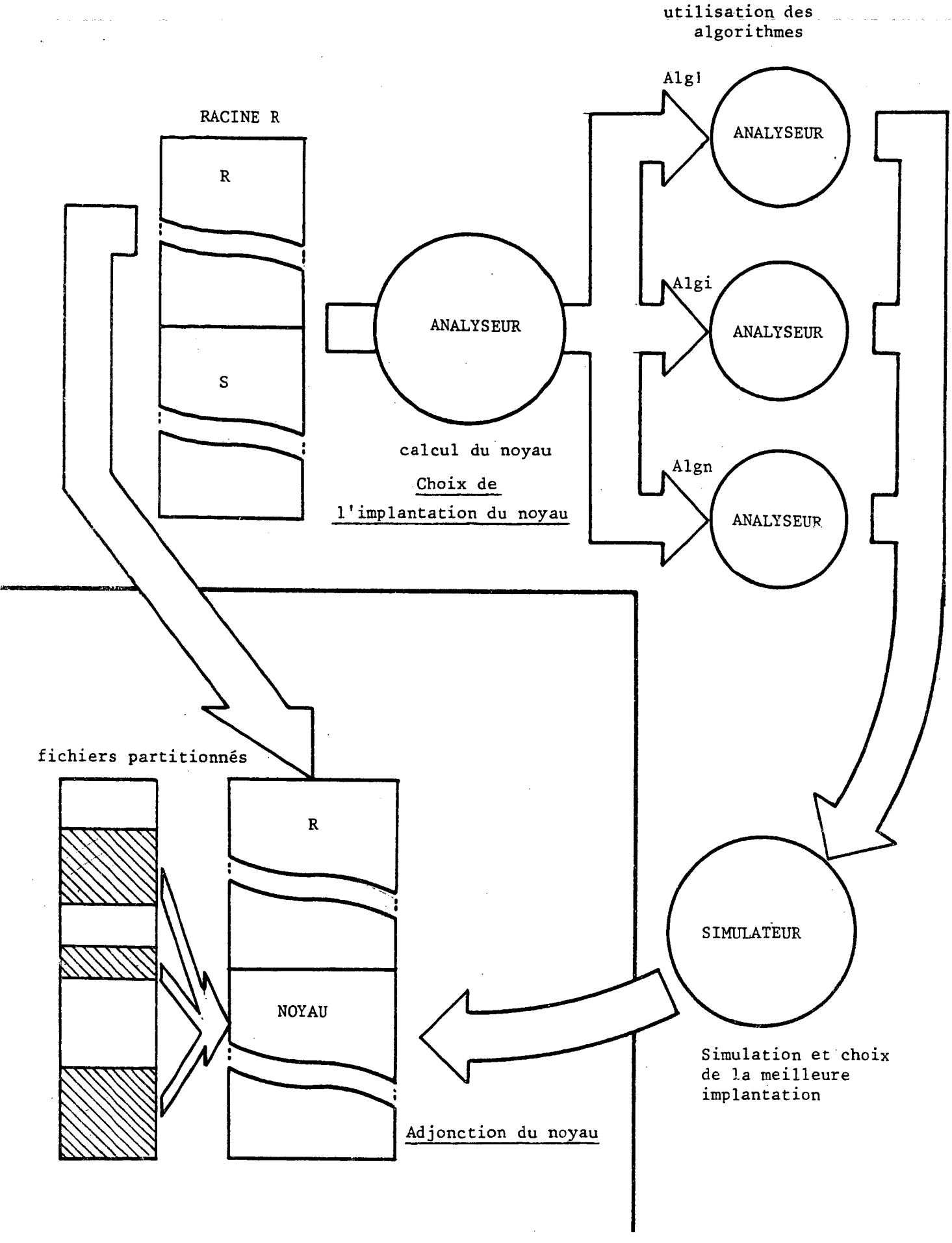

utilisation des algorithmes
Alg1
ANALYSEUR
RACINE R
R
S
ANALYSEUR
Algi
ANALYSEUR
calcul du noyau
Choix de
l'implantation du noyau
Algn
ANALYSEUR
fichiers partitionnés
R
SIMULATEUR
NOYAU
Simulation et choix
de la meilleure
implantation
Adjonction du noyau

Figure 14

CHAPITRE 4 : RESULTATS

Nous utilisons ici les résultats qui ont été obtenus sur la bibliothèque la plus accédée au centre de calcul de l'INRIA : LIBREL (1.3.2).

4.1 Comparaison des algorithmes

La figure 15 représente les algorithmes PM, CH et ND en fonction de CN : en abscisse le nombre de pages du noyau, en ordonnée le % d'accès disque supplémentaire.

Il apparaît que :

- les différences sont faibles (au plus de l'ordre de 10%),
- l'algorithme CN est apparu comme le plus performant en règle générale,
- l'algorithme PM donne ici de moins bons résultats que CN. Cependant l'alignement sur des frontières de quantums a amélioré légèrement les résultats dans le cas d'autres bibliothèques (5%).

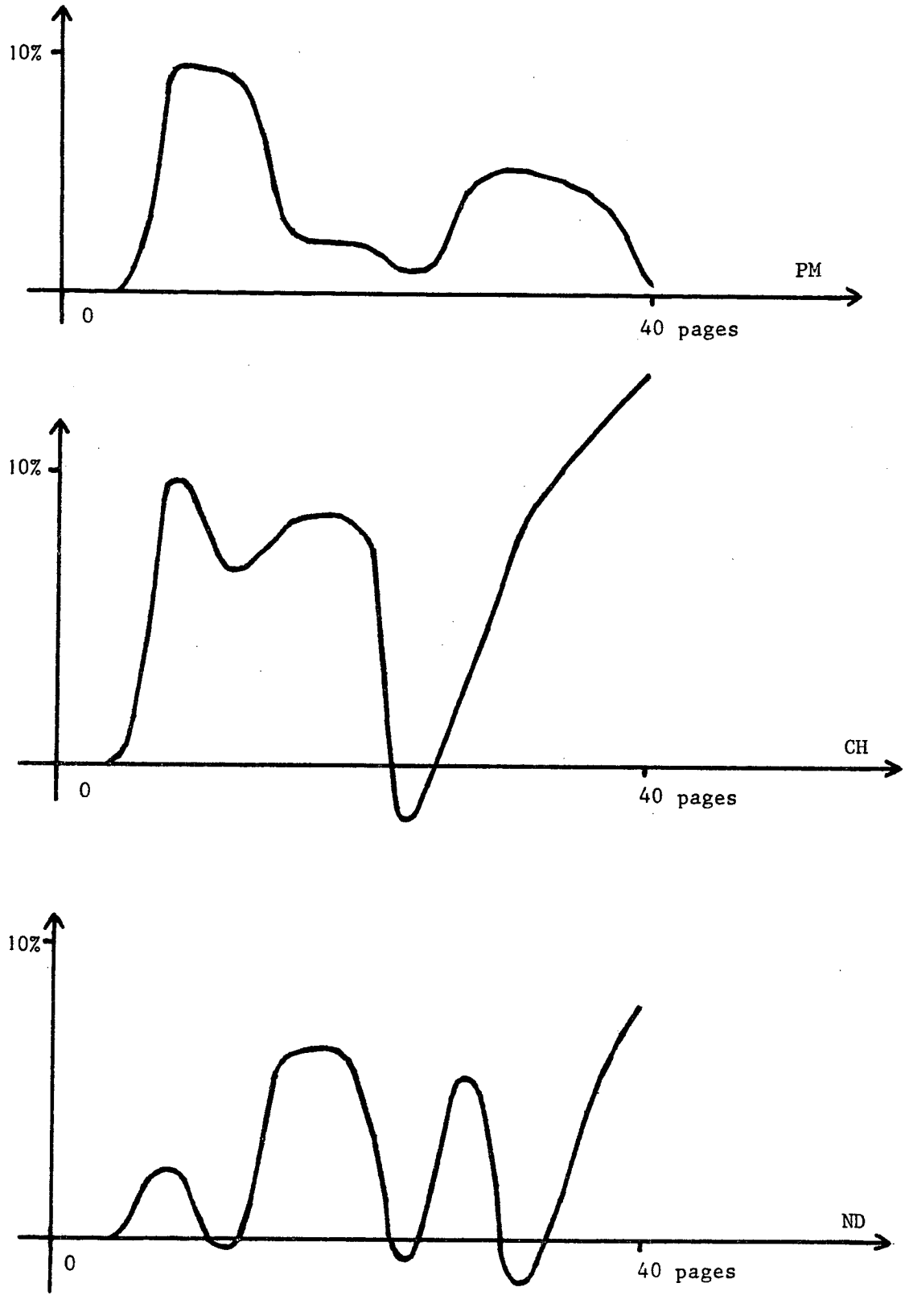
Il faut noter le faible gain apporté pour des algorithmes sophistiqués par rapport à un algorithme simple comme CN.

4.2 Visualisation intermédiaire

Rappelons qu'il s'agit des courbes obtenues lors de la simulation intermédiaire (3.5.2) : ces courbes sont basées sur un Noyau organisé selon l'algorithme CN (3.3.2).

Les courbes des figures 12 et 13 sont donc une bonne représentation de l'utilisation de la bibliothèque en fonction de la taille du Noyau.

Remarquons qu'un noyau de faible taille permet des améliorations importantes : un noyau égal à 10% de la taille de la bibliothèque permet des gains de l'ordre de 40 à 50% en accès disque et en transfert d'octets.

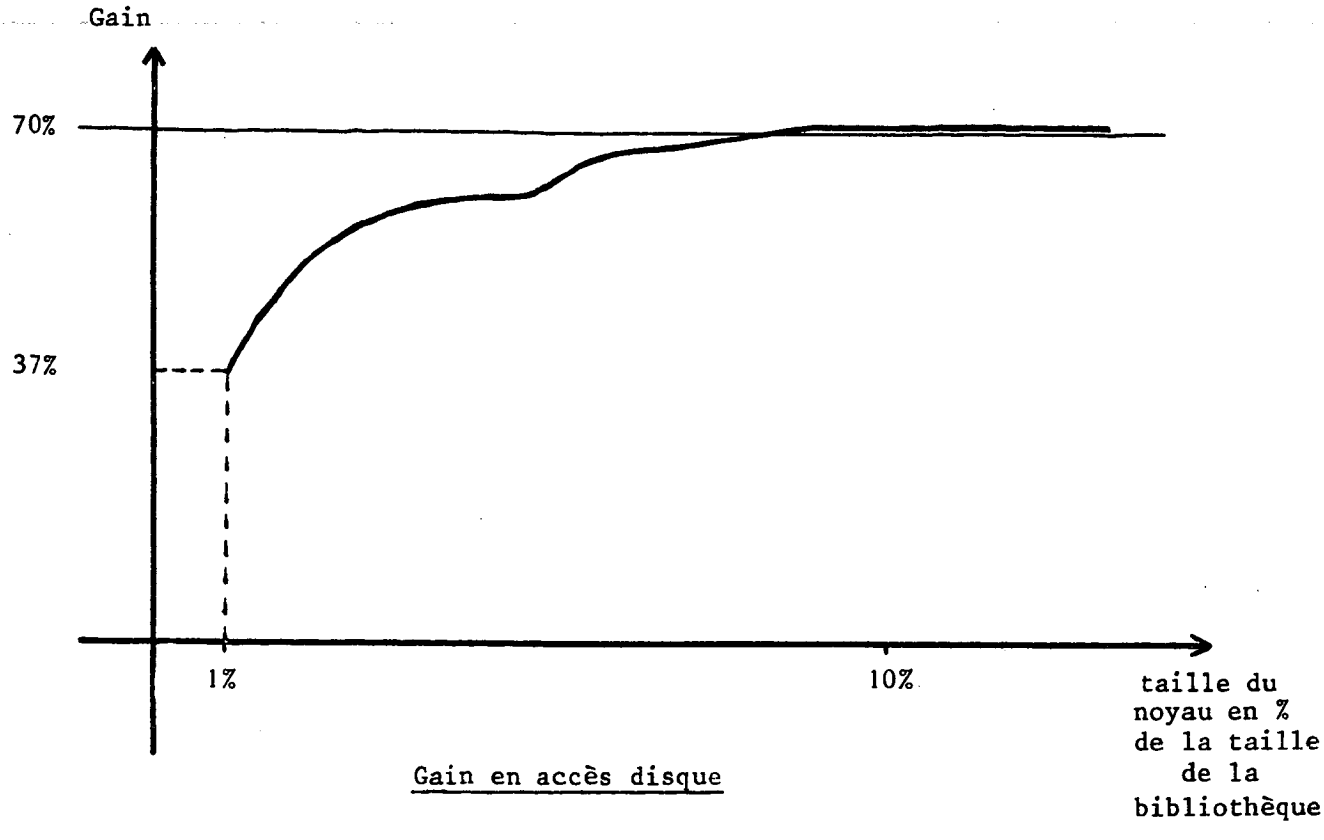


Comparaison des algorithmes

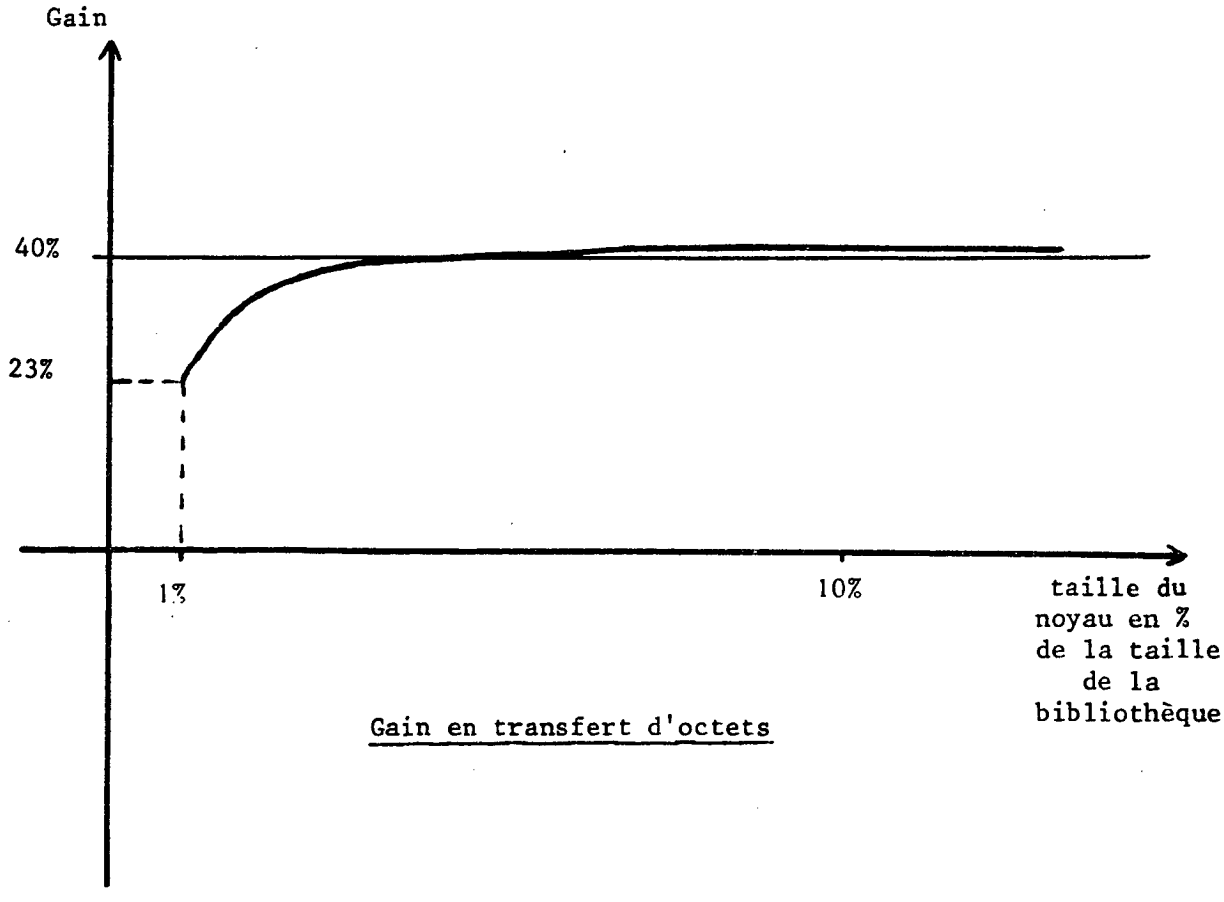
Figure 15

4.3 Améliorations apportées par la restructuration

Les courbes de visualisation (4.2) ne tiennent pas compte du chargement en mémoire centrale de la RACINE. En tenant compte de ce chargement nous obtenons des courbes de gain par rapport à la structure partitionnée telle qu'elle est utilisée par le processeur standard du constructeur (figure 16).



Gain en accès disque



Gain en transfert d'octets

Figure 16

CONCLUSION

Les bibliothèques de sous-programmes figurent parmi les logiciels indispensables d'un système informatique. Devant l'importance des problèmes posés par la gestion des milliers d'éléments prédéfinis qui les constituent, nous avons commencé par en visualiser la structure sous la forme d'une représentation matricielle des liens inter-modules.

Nous avons poursuivi l'étude de cette structure par la mise en oeuvre d'algorithmes permettant d'y détecter certaines anomalies : cycles, trop grande interconnexion des modules, etc...

Nous avons ensuite mesuré les fréquences d'utilisation des modules et mis ainsi en évidence une grande disparité dans ces utilisations : une faible proportion de la bibliothèque (Noyau), en nombre de modules (10%) et en taille (15%), concentre la plus grande partie de l'utilisation (85%).

Nous avons appliqué à ce noyau les algorithmes de regroupement développés dans le cadre du système OPALE, ce qui nous a permis d'obtenir un gain moyen de 70% sur les accès disque effectués lors de la consultation d'une bibliothèque.

La comparaison des résultats obtenus a fait apparaître que les algorithmes sophistiqués qui semblaient, intuitivement, devoir donner de meilleurs résultats, ne surpassent que rarement des algorithmes plus simples.

Nous allons pouvoir vérifier ces tendances par la mise à la disposition de tous les utilisateurs de SIRIS8 du système que nous avons réalisé.

Il est en effet partie intégrante du système OPALE, dont la fiabilité et l'efficacité ont abouti à ce que les utilisateurs en demandent et obtiennent en mai 1980 la mise au catalogue général de CII-HB.

BIBLIOGRAPHIE

ACH76

ACHARD M.S., BABONNEAU J-Y., MORISSET G.,
"Adaptation automatique des programmes au milieu paginé"
Rapport de recherche LABORIA numéro 196, octobre 1976.

ACH78

ACHARD M.S., BABONNEAU J.Y., CARPENTIER M., MORISSET G.,
"Clustering Algorithms in the OPALE Restructuring System"
International Conference on the Performance of Computer Installations.
22-23 juin 1978 Gordone Riviera - Italie

BAB77

BABONNEAU J.Y., CARPENTIER M., MORISSET G.,
"Automatic and general solution to the adaptation of programs in a paging environment"
Proceeding of 6th ACM Symposium an Operating Systems Principles
novembre 1977 - pages 109-116.

DID71

DIDAY E.,
"Une nouvelle méthode en classification automatique et reconnaissance des formes : la méthode des nuées dynamiques"
Revue de statistique appliquée - Vol XIX numéro 2 - 1971.

MOR75

MORISSET G.,
"Adaptation automatique des programmes au milieu paginé"
Thèse de 3ème cycle, Paris VI, juin 1975.

PRE72

PRESSER L., WHITE J.R.,
"Linkers and loaders"
Computing Surveys - Vol 4, numéro 3 (septembre 1972).

SGF75

"Système de gestion de fichiers, SGF sous SIRIS 7/8"
Réf 4498 E/FR. CII-HB.